

研究室ゼミ

# 住井研究室の ステキな Beamer テンプレート

ラムダ 小太郎

東北大学大学院情報科学研究科  
住井・松田研究室

2000 年 1 月 1 日

# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
- 4 レイアウト

# アウトライン

- 1 単純な使い方  
    フォント  
    箇条書き
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
- 4 レイアウト

# アウトライン

- 1 単純な使い方  
    フォント  
    箇条書き
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
- 4 レイアウト

# フォント

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

こんにちは、世界。

# アウトライン

- 1 単純な使い方
  - フォント
  - 箇条書き
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
- 4 レイアウト

番号なし箇条書き：

- 項目 1
- 項目 2
- 項目 3

番号つき箇条書き：

1. 項目 1
2. 項目 2
3. 項目 3

# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能  
ブロック  
オーバーレイ
- 3 ソースコードの書き方
- 4 レイアウト



# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能  
ブロック  
オーバーレイ
- 3 ソースコードの書き方
- 4 レイアウト

# ブロックの使用例

## ブロックのタイトル

ブロックの内容。

## ブロックのタイトル

exampleblock は例のためのブロックです。

## ブロックのタイトル

alertblock は強調のためのブロックです。alert のブロック版だと思えばいいでしょう。

# 定理環境の使用例

## 定義 1.1 (定義のタイトル)

定義の内容

## 補題 2.2 (補題のタイトル)

補題の内容

## 定理 3.4 (定理のタイトル)

定理の内容

## 証明のタイトル.

証明の内容



# ブロック環境

次の環境が使えます。

- block
- exampleblock
- alertblock
- 定義 (definition)
- 公理 (axiom)
- 定理 (theorem)
- 補題 (lemma)
- 系 (corollary)
- 命題 (proposition)
- 証明 (proof) — 他の環境と少しだけ使い方が違うので注意

※ block, exampleblock, alertblock はただの色違い。それ以外は block 環境と同じ色。

# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能  
ブロック  
オーバーレイ
- 3 ソースコードの書き方
- 4 レイアウト

# オーバーレイ

オーバーレイ (overlay) とは、

# オーバーレイ

オーバーレイ (overlay) とは、

- 単純なアニメーションみたいなもの

# オーバーレイ

オーバーレイ (overlay) とは、

- 単純なアニメーションみたいなもの
- 最初のスライドでは隠していた文字や図形を、あとから表示させる



オーバーレイ (overlay) とは、

- 単純なアニメーションみたいなもの
- 最初のスライドでは隠していた文字や図形を、あとから表示させる
- よく使うのは pause (他にもいろいろある)

# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
  - 一時的にスタイル or 言語を変更する
  - ソースコードの手動強調表示
- 4 レイアウト

# ソースコードの書き方

ソースコードは `verbatim` 環境でも書けるが、あまり綺麗ではない。

`listings` を使うのがおすすめ：

- `listings.sty` — LaTeX で綺麗なソースコードを書くためのスタイルファイル
- `jlisting.sty` — ソースコード中で日本語を使いたい時に必要（`listings.sty` と併用）

# ソースコードの書き方

- frame 環境のオプションに fragile を指定する
  - 指定の方法はソースコードを参照
  - 指定しないと、コンパイルできない
- listings はあまり高度な自動色付けができない
  - せいぜい、予約語の強調とか、文字列・コメントの色つけ程度
  - 細かい強調は手動で行ったほうが良い（後述）

# ソースコードの例

- 長いソースコードには `lstlisting` 環境を使う
- 文中のソースコードには `lstinline` マクロを使う（用法は `verb` と同じ）

例 1) `lstlisting` 環境：

```
type 'a bin_tree =  
  | Leaf of 'a  
  | Node of 'a bin_tree * 'a bin_tree  
  
let rec listup_nodes = function  
  | Leaf x -> [x]  
  | Node (r, l) -> (listup_nodes r) @ (listup_nodes l)
```

例 2) `lstinline` マクロ：

`listup_nodes` の型は `'a bin_tree -> 'a list` である。

# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
  - 一時的にスタイル or 言語を変更する
  - ソースコードの手動強調表示
- 4 レイアウト

# 一時的にスタイル or 言語を変更する

ソースコードの強調表示の設定：

- 共通の定義はプリアンプルの `lstset` で行う。
- 個別に変更するときは、`lstlisting`、`lstinline` のオプションで指定する。

例 1) フレームなし

```
let rec fact n =  
  if n = 0 then 1 else n * (fact (n - 1))
```

例 2) C 言語に変更

```
int fact (int n) {  
  if (n == 0) {  
    return 1;  
  } else {  
    return n * fact(n - 1);  
  }  
}
```

# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方**
  - 一時的にスタイル or 言語を変更する
  - ソースコードの手動強調表示
- 4 レイアウト



# ソースコードの手動強調表示

以下の書式で強調表示ができるようになっている。  
( 使い方はソースコードを参照 )

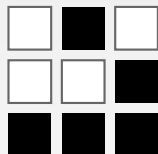
- @/.../@ — イタリック : *hoge*
- @r{...}@ — 赤 : **hoge**
- @g{...}@ — 緑 : **hoge**
- @b{...}@ — 青 : **hoge**

例 )

```
let fact n
  let rec fact' i acc =
    if i = 0 then acc else fact' (i - 1) (n * acc)
  in
  fact' n 1
```

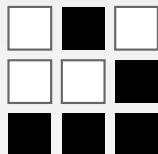
# アウトライン

- 1 単純な使い方
- 2 ちょっと特殊な機能
- 3 ソースコードの書き方
- 4 レイアウト**



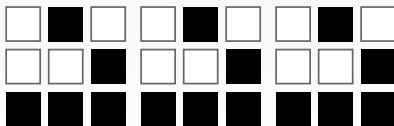
- ページを横に分割
- 図・表・文を横に並べて配置可能
- よく使うレイアウト
- minipage 環境でも同じ事ができる

# columns/column 環境



- ページを横に分割
- 図・表・文を横に並べて配置可能
- よく使うレイアウト
- minipage 環境でも同じ事ができる

入れ子にしてみる



- 3 つ以上の分割も可能
- 入れ子も可能
- 柔軟に使えて便利！

# APPENDIX

## 5 予備のスライド

# 予備のスライド

予備スライドは `appendix` 環境の中に書きましょう。